C interfaces to GALAHAD CRO

Jari Fowkes and Nick Gould
STFC Rutherford Appleton Laboratory
Thu Jun 22 2023

Chapter 1

GALAHAD C package cro

1.1 Introduction

1.1.1 Purpose

Provides a **crossover** from a solution to the **convex quadratic programming problem**

$$\text{minimize } \; q(x) = \frac{1}{2} x^T H x + g^T x + f$$

subject to the general linear constraints

$$c_i^l \leq a_i^T x \leq c_i^u, \quad i = 1, \ldots, m,$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j = 1, \ldots, n,$$

found by an interior-point method to one in which the **matrix of defining active constraints/variables is of full rank.** Here, the n by n symmetric, positive-semi-definite matrix H, the vectors g, a_i, c^l, c^u, x^l, x^u, the scalar f are given. In addition a solution x along with optimal Lagrange multipliers y for the general constraints and dual variables z for the simple bounds must be provided (see Section∼galmethod}). These will be adjusted as necessary. Any of the constraint bounds c_i^l, c_i^u, x_j^l and x_j^u may be infinite. Full advantage is taken of any zero coefficients in the matrix H or the matrix A of vectors a_i.

1.1.2 Authors

N. I. M. Gould, STFC-Rutherford Appleton Laboratory, England.

C interface, additionally J. Fowkes, STFC-Rutherford Appleton Laboratory.

Julia interface, additionally A. Montoison and D. Orban, Polytechnique Montréal.

1.1.3 Originally released

August 2010, C interface January 2022.

1.1.4 Terminology

Any required solution x necessarily satisfies the primal optimality conditions

$$Ax = c \tag{1a}$$

and

$$c^l \leq c \leq c^u, \;\; x^l \leq x \leq x^u, \tag{1b}$$

the dual optimality conditions

$$Hx + g = A^T y + z \tag{2a}$$

where

$$y = y^l + y^u, \;\; z = z^l + z^u, \;\; y^l \geq 0, \;\; y^u \leq 0, \;\; z^l \geq 0 \;\text{ and }\; z^u \leq 0, \tag{2b}$$

and the complementary slackness conditions

$$(Ax - c^l)^T y^l = 0, \;\; (Ax - c^u)^T y^u = 0, \;\; (x - x^l)^T z^l = 0 \;\text{ and }\; (x - x^u)^T z^u = 0, \tag{3}$$

where the vectors y and z are known as the Lagrange multipliers for the general linear constraints, and the dual variables for the bounds, respectively, and where the vector inequalities hold component-wise.

1.1.5 Method

Denote the active constraints by $A_A x = c_A$ and the active bounds by $I_A x = x_A$. Then any optimal solution satisfies the linear system

$$\begin{pmatrix} H & -A_A^T & -I_A^T \\ A_A & 0 & 0 \\ I_A & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y_A \\ z_A \end{pmatrix} = \begin{pmatrix} -g \\ c_A \\ x_A \end{pmatrix}.$$

where y_A and z_A are the corresponding active Lagrange multipliers and dual variables respectively. Consequently the difference between any two solutions $(\Delta x, \Delta y, \Delta z)$ must satisfy

$$\begin{pmatrix} H & -A_A^T & -I_A^T \\ A_A & 0 & 0 \\ I_A & 0 & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y_A \\ \Delta z_A \end{pmatrix} = 0. \tag{4}$$

Thus there can only be multiple solution if the coefficient matrix K of (4) is singular. The algorithm used in CRO exploits this. The matrix K is checked for singularity using the GALAHAD package ULS. If K is non singular, the solution is unique and the solution input by the user provides a linearly independent active set. Otherwise K is singular, and partitions $A_A^T = (A_B^T \;\; A_N^T)$ and $I_A^T = (I_B^T \;\; I_N^T)$ are found so that

$$\begin{pmatrix} H & -A_B^T & -I_B^T \\ A_B & 0 & 0 \\ I_B & 0 & 0 \end{pmatrix}$$

is non-singular and the **non-basic** constraints A_N^T and I_N^T are linearly dependent on the **basic** ones $(A_B^T \;\; I_B^T)$. In this case (4) is equivalent to

$$\begin{pmatrix} H & -A_B^T & -I_B^T \\ A_B & 0 & 0 \\ I_B & 0 & 0 \end{pmatrix} = \begin{pmatrix} A_N^T \\ 0 \\ 0 \end{pmatrix} \Delta y_N + \begin{pmatrix} I_N^T \\ 0 \\ 0 \end{pmatrix} \Delta z_N \tag{5}$$

Thus, starting from the user's (x, y, z) and with a factorization of the coefficient matrix of (5) found by the GALAHAD package SLS, the alternative solution $(x + \alpha x, y + \alpha y, z + \alpha z)$, featuring $(\Delta x, \Delta y_B, \Delta z_B)$ from (5) in which successively one of the components of Δy_N and Δz_N in turn is non zero, is taken. The scalar α at each stage is chosen to be the largest possible that guarantees (2.b); this may happen when a non-basic multiplier/dual variable reaches zero, in which case the corresponding constraint is disregarded, or when this happens for a basic multiplier/dual variable, in which case this constraint is exchanged with the non-basic one under consideration and disregarded. The latter corresponds to changing the basic-non-basic partition in (5), and subsequent solutions may be found by updating the factorization of the coefficient matrix in (5) following the basic-non-basic swap using the GALAHAD package SCU.

1.1.6 Reference

1.1.7 Call order

To solve a given problem, functions from the cro package must be called in the following order:

- cro_initialize - provide default control parameters and set up initial data structures
- cro_read_specfile (optional) - override control values by reading replacement values from a file
- cro_crossover_solution - move from a primal-dual soution to a full rank one
- cro_terminate - deallocate data structures

See Section **??** for examples of use.

1.1.8 Array indexing

Both C-style (0 based) and fortran-style (1-based) indexing is allowed. Choose `control.f_indexing` as `false` for C style and `true` for fortran style; add 1 to input integer arrays if fortran-style indexing is used, and beware that return integer arrays will adhere to this.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

File Documentation

3.1 galahad_cro.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_sls.h"
#include "galahad_sbls.h"
#include "galahad_uls.h"
#include "galahad_ir.h"
#include "galahad_scu.h"
```

Data Structures

- struct cro_control_type
- struct cro_time_type
- struct cro_inform_type

Functions

- void cro_initialize (void **data, struct cro_control_type *control, int *status)
- void cro_read_specfile (struct cro_control_type *control, const char specfile[ ])
- void cro_crossover_solution (void **data, struct cro_control_type *control, struct cro_inform_type *inform, int n, int m, int m_equal, int h_ne, const real_wp_ H_val[ ], const int H_col[ ], const int H_ptr[ ], int a_ne, const real_wp_ A_val[ ], const int A_col[ ], const int A_ptr[ ], const real_wp_ g[ ], const real_wp_ c_l[ ], const real_wp_ c_u[ ], const real_wp_ x_l[ ], const real_wp_ x_u[ ], real_wp_ x[ ], real_wp_ c[ ], real_wp_ y[ ], real_wp_ z[ ], int x_stat[ ], int c_stat[ ])
- void cro_terminate (void **data, struct cro_control_type *control, struct cro_inform_type *inform)

3.1.1 Data Structure Documentation

3.1.1.1 struct cro_control_type

control derived type as a C struct

Examples

crot.c, and crotf.c.

Data Fields

bool	f_indexing	use C or Fortran sparse matrix indexing
int	error	error and warning diagnostics occur on stream error
int	out	general output occurs on stream out
int	print_level	the level of output required is specified by print_level
int	max_schur_complement	the maximum permitted size of the Schur complement before a refactorization is performed
real_wp_	infinity	any bound larger than infinity in modulus will be regarded as infinite
real_wp_	feasibility_tolerance	feasibility tolerance for KKT violation
bool	check_io	if .check_io is true, the input (x,y,z) will be fully tested for consistency
bool	refine_solution	if .refine solution is true, attempt to satisfy the KKT conditions as accurately as possible
bool	space_critical	if .space_critical is true, every effort will be made to use as little space as possible. This may result in longer computation time
bool	deallocate_error_fatal	if .deallocate_error_fatal is true, any array/pointer deallocation error will terminate execution. Otherwise, computation will continue
char	symmetric_linear_solver[31]	indefinite linear equation solver
char	unsymmetric_linear_solver[31]	unsymmetric linear equation solver
char	prefix[31]	all output lines will be prefixed by .prefix(2:LEN(TRIM(.prefix))-1) where .prefix contains the required string enclosed in quotes, e.g. "string" or 'string'
struct sls_control_type	sls_control	control parameters for SLS
struct sbls_control_type	sbls_control	control parameters for SBLS
struct uls_control_type	uls_control	control parameters for ULS
struct ir_control_type	ir_control	control parameters for iterative refinement

3.1.1.2 struct cro_time_type

time derived type as a C struct

Data Fields

real_sp_	total	the total CPU time spent in the package
real_sp_	analyse	the CPU time spent reordering the matrix prior to factorization
real_sp_	factorize	the CPU time spent factorizing the required matrices
real_sp_	solve	the CPU time spent computing corrections
real_wp_	clock_total	the total clock time spent in the package
real_wp_	clock_analyse	the clock time spent analysing the required matrices prior to factorizat
real_wp_	clock_factorize	the clock time spent factorizing the required matrices
real_wp_	clock_solve	the clock time spent computing corrections

3.1.1.3 struct cro_inform_type

inform derived type as a C struct

Examples

crot.c, and crotf.c.

Data Fields

int	status	return status. See CRO_solve for details
int	alloc_status	the status of the last attempted allocation/deallocation
char	bad_alloc[81]	the name of the array for which an allocation/deallocation error occurred
int	dependent	the number of dependent active constraints
struct cro_time_type	time	timings (see above)
struct sls_inform_type	sls_inform	information from SLS
struct sbls_inform_type	sbls_inform	information from SBLS
struct uls_inform_type	uls_inform	information from ULS
int	scu_status	information from SCU
struct scu_inform_type	scu_inform	see scu_status
struct ir_inform_type	ir_inform	information from IR

3.1.2 Function Documentation

3.1.2.1 cro_initialize()

```
void cro_initialize (
             void ** data,
             struct cro_control_type * control,
             int * status )
```

Set default control values and initialize private data

Parameters

in,out	*data*	holds private internal data
out	*control*	is a struct containing control information (see cro_control_type)
out	*status*	is a scalar variable of type int, that gives the exit status from the package. Possible values are (currently): • 0. The initialization was succesful.

Examples

crot.c, and crotf.c.

3.1.2.2 cro_read_specfile()

```
void cro_read_specfile (
             struct cro_control_type * control,
             const char specfile[] )
```

Read the content of a specification file, and assign values associated with given keywords to the corresponding control parameters. By default, the spcification file will be named RUNCRO.SPC and lie in the current directory. Refer to Table 2.1 in the fortran documentation provided in $GALAHAD/doc/cro.pdf for a list of keywords that may be set.

Parameters

in,out	*control*	is a struct containing control information (see cro_control_type)
in	*specfile*	is a character string containing the name of the specification file

3.1.2.3 cro_crossover_solution()

```
void cro_crossover_solution (
             void ** data,
             struct cro_control_type * control,
             struct cro_inform_type * inform,
             int n,
             int m,
             int m_equal,
             int h_ne,
             const real_wp_ H_val[],
             const int H_col[],
             const int H_ptr[],
             int a_ne,
             const real_wp_ A_val[],
             const int A_col[],
             const int A_ptr[],
             const real_wp_ g[],
             const real_wp_ c_l[],
             const real_wp_ c_u[],
             const real_wp_ x_l[],
             const real_wp_ x_u[],
             real_wp_ x[],
             real_wp_ c[],
             real_wp_ y[],
             real_wp_ z[],
             int x_stat[],
             int c_stat[] )
```

Crosover the solution from a primal-dual to a basic one.

Parameters

in	*control*	is a struct whose members provide control paramters for the remaining prcedures (see cro_control_type). The parameter .status is as follows:
in,out	*data*	holds private internal data.
out	*inform*	is a struct containing output information (see cro_inform_type). The component .status gives the exit status from the package. Possible values are: • 0. The crossover was succesful. • -1. An allocation error occurred. A message indicating the offending array is written on unit control.error, and the returned allocation status and a string containing the name of the offending array are held in inform.alloc_status and inform.bad_alloc respectively. • -2. A deallocation error occurred. A message indicating the offending array is written on unit control.error and the returned allocation status and a string containing the name of the offending array are held in inform.alloc_status and inform.bad_alloc respectively. • -3. The restrictions n $>$ 0 or m $>=$ m_equal $>=$ 0 has been violated. • -4 the bound constraints are inconsistent. • -5 the general constraints are likely inconsistent. • -9 an error has occured in SLS_analyse. • -10 an error has occured in SLS_factorize. • -11 an error has occured in SLS_solve. • -12 an error has occured in ULS_factorize. • -14 an error has occured in ULS_solve. • -16 the residuals are large; the factorization may be unsatisfactory.
in	*n*	is a scalar variable of type int, that holds the number of variables.
in	*m*	is a scalar variable of type int, that holds the number of general linear constraints.
in	*m_equal*	is a scalar variable of type int, that holds the number of general linear equality constraints. Such constraints must occur first in A.
in	*h_ne*	is a scalar variable of type int, that holds the number of entries in the **lower triangular** part of the Hessian matrix H.
in	*H_val*	is a one-dimensional array of type double, that holds the values of the entries of the lower triangular part of the Hessian matrix H. The entries are stored by consecutive rows, the order within each row is unimportant.
in	*H_col*	is a one-dimensional array of type int, that holds the column indices of the lower triangular part of H, in the same order as those in H_val.
in	*H_ptr*	is a one-dimensional array of size n+1 and type int, that holds the starting position of each row of the lower triangular part of H. The n+1-st component holds the total number of entries (plus one if fortran indexing is used).
in	*a_ne*	is a scalar variable of type int, that holds the number of entries in the constraint Jacobian matrix A.
in	*A_val*	is a one-dimensional array of type double, that holds the values of the entries of the constraint Jacobian matrix A. The entries are stored by consecutive rows, the order within each row is unimportant. **Equality constraints must be ordered first.**
in	*A_col*	is a one-dimensional array of size A_ne and type int, that holds the column indices of A in the same order as those in A_val.

Parameters

in	*A_ptr*	is a one-dimensional array of size m+1 and type int, that holds the starting position of each row of A. The m+1-st component holds the total number of entries (plus one if fortran indexing is used).
in	*g*	is a one-dimensional array of size n and type double, that holds the linear term g of the objective function. The j-th component of g, j = 0, ... , n-1, contains g_j.
in	*c_l*	is a one-dimensional array of size m and type double, that holds the lower bounds c^l on the constraints Ax. The i-th component of c_l, i = 0, ... , m-1, contains c_i^l.
in	*c_u*	is a one-dimensional array of size m and type double, that holds the upper bounds c^l on the constraints Ax. The i-th component of c_u, i = 0, ... , m-1, contains c_i^u.
in	*x_l*	is a one-dimensional array of size n and type double, that holds the lower bounds x^l on the variables x. The j-th component of x_l, j = 0, ... , n-1, contains x_j^l.
in	*x_u*	is a one-dimensional array of size n and type double, that holds the upper bounds x^l on the variables x. The j-th component of x_u, j = 0, ... , n-1, contains x_j^l.
in,out	*x*	is a one-dimensional array of size n and type double, that holds the values x of the optimization variables. The j-th component of x, j = 0, ... , n-1, contains x_j.
in,out	*c*	is a one-dimensional array of size m and type double, that holds the residual $c(x) = Ax$. The i-th component of c, j = 0, ... , n-1, contains $c_j(x)$.
in,out	*y*	is a one-dimensional array of size n and type double, that holds the values y of the Lagrange multipliers for the general linear constraints. The j-th component of y, j = 0, ... , n-1, contains y_j.
in,out	*z*	is a one-dimensional array of size n and type double, that holds the values z of the dual variables. The j-th component of z, j = 0, ... , n-1, contains z_j.
in,out	*x_stat*	is a one-dimensional array of size n and type int, that must be set on entry to give the status of the problem variables. If x_stat(j) is negative, the variable x_j is active on its lower bound, if it is positive, it is active and lies on its upper bound, and if it is zero, it is inactiive and lies between its bounds. On exit, the j-th component of x_stat is -1 if the variable is basic and active on its lower bound, -2 it is non-basic but active on its lower bound, 1 if it is basic and active on its upper bound, 2 it is non-basic but active on its upper bound, and 0 if it is inactive.
in,out	*c_stat*	is a one-dimensional array of size m and type int, that must be set on entry to give the status of the general linear constraints. If c_stat(i) is negative, the constraint value $a_i^T x$ is active on its lower bound, if it is positive, it is active and lies on its upper bound, and if it is zero, it is inactiive and lies between its bounds. On exit, the i-th component of x_stat is -1 if the constraint is basic and active on its lower bound, -2 it is non-basic but active on its lower bound, 1 if it is basic and active on its upper bound, 2 it is non-basic but active on its upper bound, and 0 if it is inactive.

Examples

crot.c, and crotf.c.

3.1.2.4 cro_terminate()

```
void cro_terminate (
            void ** data,
            struct cro_control_type * control,
            struct cro_inform_type * inform )
```

Deallocate all internal private storage

Parameters

`in,out`	*data*	holds private internal data
`out`	*control*	is a struct containing control information (see cro_control_type)
`out`	*inform*	is a struct containing output information (see cro_inform_type)

Examples

crot.c, and crotf.c.

Chapter 4

Example Documentation

4.1 crot.c

This is an example of how to use the package.

```
/* crot.c */
/* Full test for the CRO C interface using C sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_cro.h"
int main(void) {
    // Derived types
    void *data;
    struct cro_control_type control;
    struct cro_inform_type inform;
    // Set problem dimensions
    int n = 11; // dimension
    int m = 3; // number of general constraints
    int m_equal = 1; // number of equality constraints
    //  describe the objective function
    int H_ne = 21;
    real_wp_ H_val[] = {1.0,0.5,1.0,0.5,1.0,0.5,1.0,0.5,1.0,0.5,
                        1.0,0.5,1.0,0.5,1.0,0.5,1.0,0.5,1.0,0.5,1.0};
    int H_col[] = {0,0,1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10};
    int H_ptr[] = {0,1,3,5,7,9,11,13,15,17,19,21};
    real_wp_ g[] = {0.5,-0.5,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-0.5};
    //  describe constraints
    int A_ne = 30;
    real_wp_ A_val[] = {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
                        1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
                        1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0};
    int A_col[] = {0,1,2,3,4,5,6,7,8,9,10,2,3,4,5,6,7,8,9,10,
                   1,2,3,4,5,6,7,8,9,10};
    int A_ptr[] = {0,11,20,30};
    real_wp_ c_l[] = {10.0,9.0,-INFINITY};
    real_wp_ c_u[] = {10.0,INFINITY,10.0};
    real_wp_ x_l[] = {0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0};
    real_wp_ x_u[]  = {INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,
                       INFINITY,INFINITY,INFINITY,INFINITY,INFINITY};
    // provide optimal variables, Lagrange multipliers and dual variables
    real_wp_ x[] = {0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0};
    real_wp_ c[] = {10.0,9.0,10.0};
    real_wp_ y[] = { -1.0,1.5,-2.0};
    real_wp_ z[] = {2.0,4.0,2.5,2.5,2.5,2.5,2.5,2.5,2.5,2.5,2.5};
    // provide interior-point constraint and variable status
    int c_stat[] = {-1,-1,1};
    int x_stat[] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
    // Set output storage
    char st;
    int status;
    printf(" C sparse matrix indexing\n\n");
    // Initialize CRO
    cro_initialize( &data, &control, &status );
    // Set user-defined control options
    control.f_indexing = false; // C sparse matrix indexing
    // crossover the solution
```

```
    cro_crossover_solution( &data, &control, &inform,
                            n, m, m_equal,
                            H_ne, H_val, H_col, H_ptr,
                            A_ne, A_val, A_col, A_ptr,
                            g, c_l, c_u, x_l, x_u, x, c, y, z,
                            x_stat, c_stat );
    printf(" CRO_crossover exit status = %1i\n", inform.status);
    // Delete internal workspace
    cro_terminate( &data, &control, &inform );
}
```

4.2 crotf.c

This is the same example, but now fortran-style indexing is used.

```
/* crotf.c */
/* Full test for the CRO C interface using C sparse matrix indexing */
#include <stdio.h>
#include <math.h>
#include "galahad_precision.h"
#include "galahad_cfunctions.h"
#include "galahad_cro.h"
int main(void) {
    // Derived types
    void *data;
    struct cro_control_type control;
    struct cro_inform_type inform;
    // Set problem dimensions
    int n = 11; // dimension
    int m = 3; // number of general constraints
    int m_equal = 1; // number of equality constraints
    //  describe the objective function
    int H_ne = 21;
    real_wp_ H_val[] = {1.0,0.5,1.0,0.5,1.0,0.5,1.0,0.5,1.0,0.5,
                        1.0,0.5,1.0,0.5,1.0,0.5,1.0,0.5,1.0,0.5,1.0};
    int H_col[] = {1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,9,10,10,11};
    int H_ptr[] = {1,2,4,6,8,10,12,14,16,18,20,22};
    real_wp_ g[] = {0.5,-0.5,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-0.5};
    //  describe constraints
    int A_ne = 30;
    real_wp_ A_val[] = {1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
                        1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,
                        1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0};
    int A_col[] = {1,2,3,4,5,6,7,8,9,10,11,3,4,5,6,7,8,9,10,11,
                   2,3,4,5,6,7,8,9,10,11 };
    int A_ptr[] = {1,12,21,31};
    real_wp_ c_l[] = {10.0,9.0,-INFINITY};
    real_wp_ c_u[] = {10.0,INFINITY,10.0};
    real_wp_ x_l[] = {0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0};
    real_wp_ x_u[]  = {INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,INFINITY,
                       INFINITY,INFINITY,INFINITY,INFINITY,INFINITY};
    // provide optimal variables, Lagrange multipliers and dual variables
    real_wp_ x[] = {0.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0};
    real_wp_ c[] = {10.0,9.0,10.0};
    real_wp_ y[] = { -1.0,1.5,-2.0};
    real_wp_ z[] = {2.0,4.0,2.5,2.5,2.5,2.5,2.5,2.5,2.5,2.5,2.5};
    // provide interior-point constraint and variable status
    int c_stat[] = {-1,-1,1};
    int x_stat[] = {-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
    // Set output storage
    char st;
    int status;
    printf(" Fortran sparse matrix indexing\n\n");
    // Initialize CRO
    cro_initialize( &data, &control, &status );
    // Set user-defined control options
    control.f_indexing = true; // Fortran sparse matrix indexing
    // crossover the solution
    cro_crossover_solution( &data, &control, &inform,
                            n, m, m_equal,
                            H_ne, H_val, H_col, H_ptr,
                            A_ne, A_val, A_col, A_ptr,
                            g, c_l, c_u, x_l, x_u, x, c, y, z,
                            x_stat, c_stat );
    printf(" CRO_crossover exit status = %1i\n", inform.status);
    // Delete internal workspace
    cro_terminate( &data, &control, &inform );
}
```